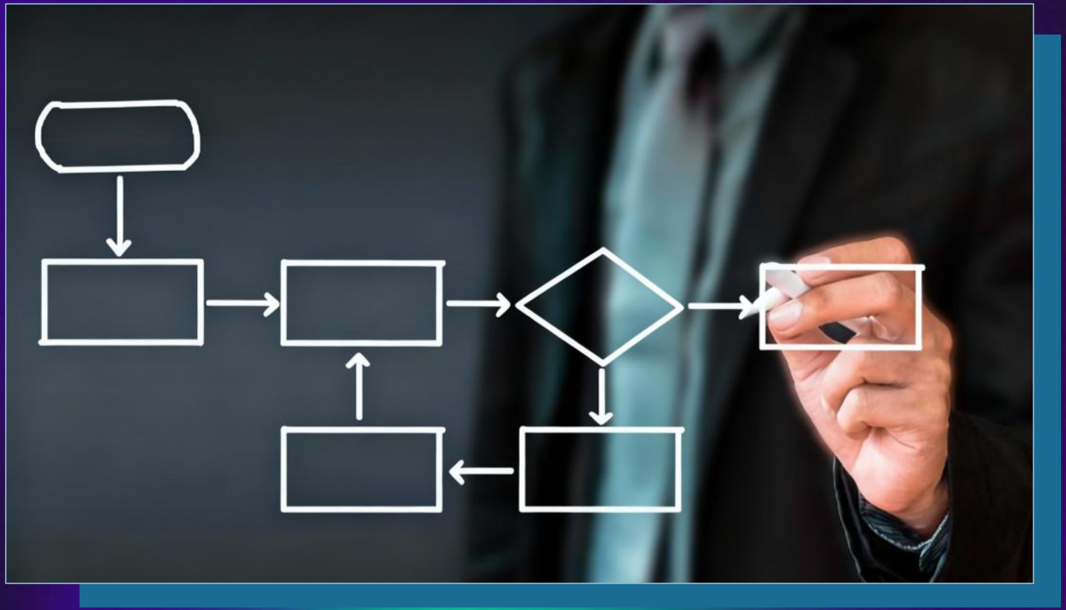
Control-Flow Logic – Part 2

Conditions, If-Else, Switch-Case, Code Blocks, Loops (for, while, do-while), Nested Loops



Svetlin Nakov, PhD

Co-founder @ SoftUni

SoftUni AI

https://ai.softuni.bg

Agenda

Sli.do Code

#AI-Programming

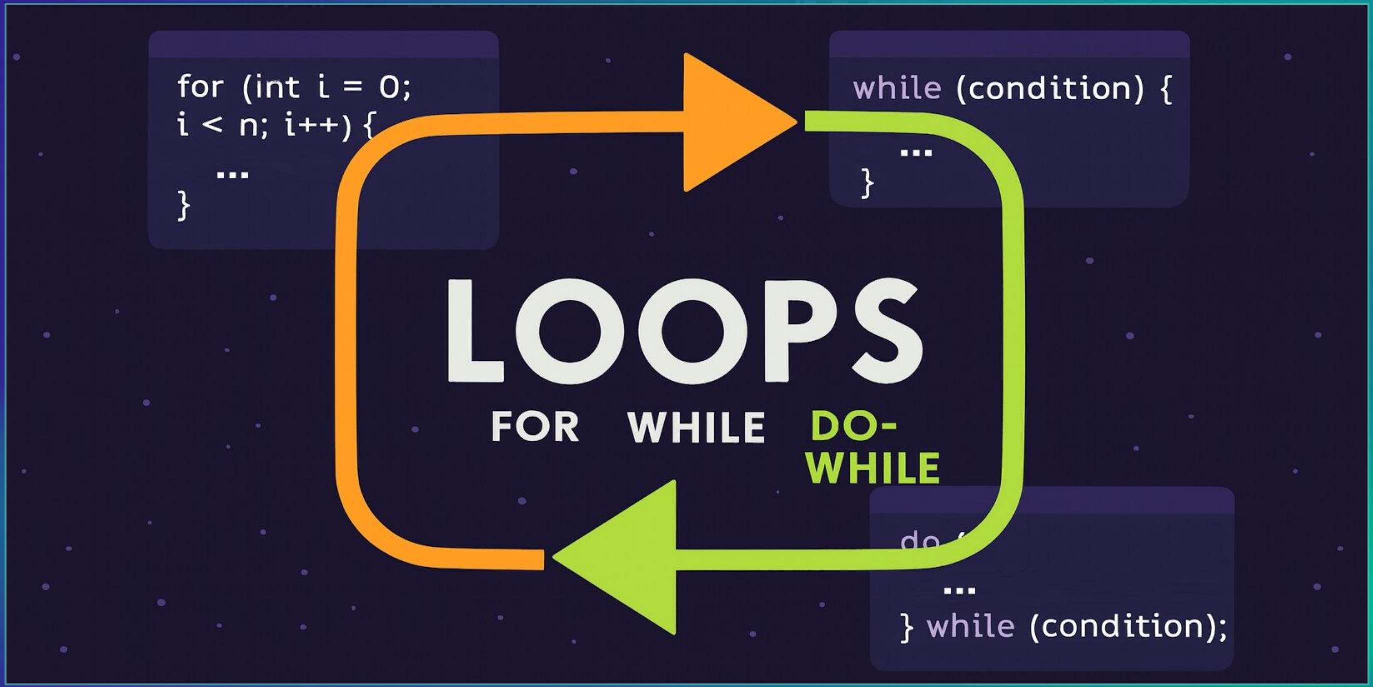
Join at

slido.com

#AI-Programming

Loops: Running a Block of Code Multiple Times

for-loop, while-loop, do-while-loop



```
for (int i = 0;
i < n; i++) {
    ...
}
```

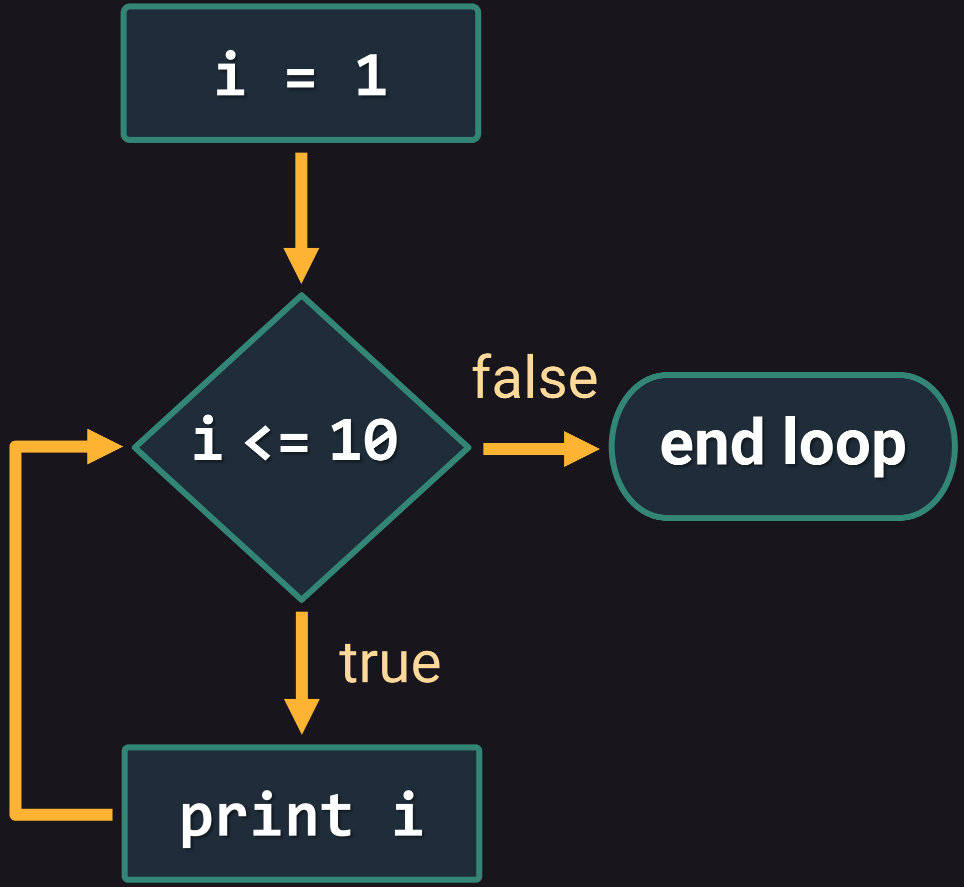
```
while (condition) {
    ...
}
```

```
do {
    ...
} while (condition);
```

What is a "Loop"?

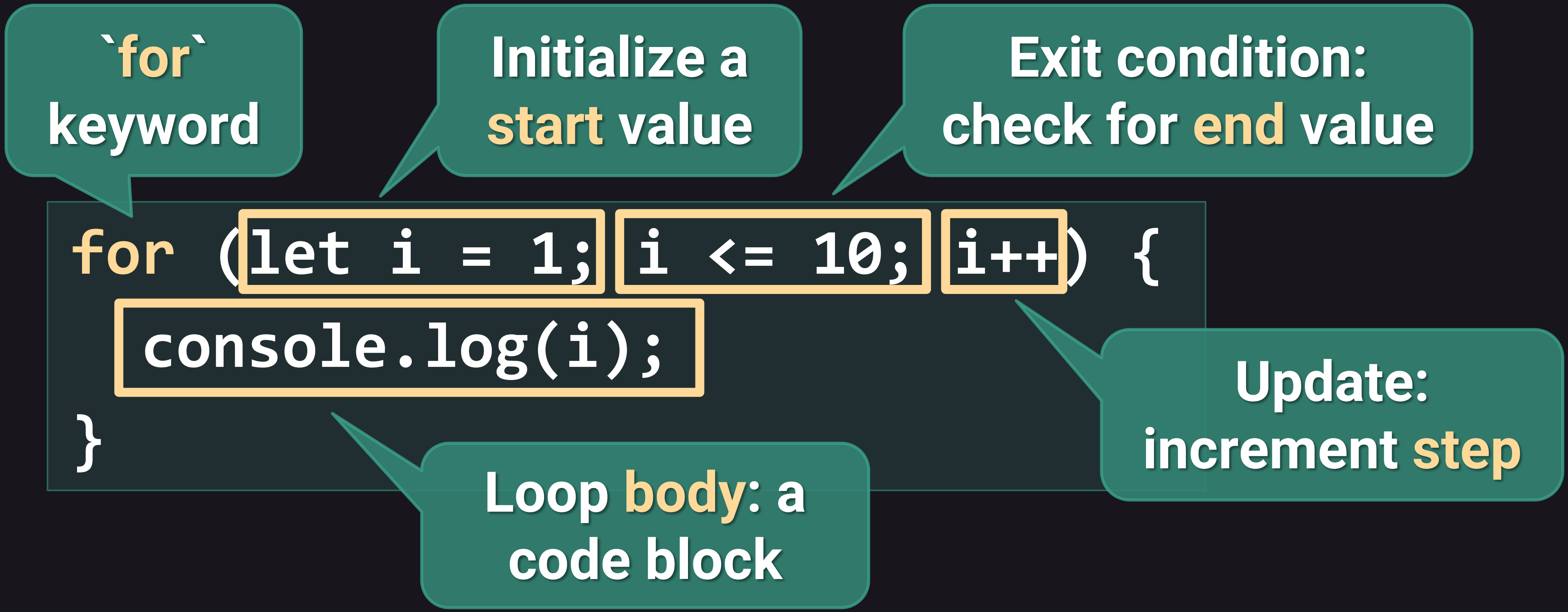
- In coding, **loops repeat a block of code** multiple times
  - **Fixed** number of times, e. g. 10
  - Until an **exit condition** happens
- Example:
  - Print the **numbers 1, 2, …, 10**:

```
for (let i = 1; i <= 10; i++) {
  console.log(i);
}
```



For-Loop Statement

- **For-loops** repeat a block of code from **start** to **end** value



```
for (let i = 1; i <= 10; i++) {
  console.log(i);
}
```

Numbers 1…100 and Their Sum

- Write a JavaScript function to print the **numbers 1…100**, along with **their cumulative sums**
  - Expected output:

```
num = 1, sum = 1
num = 2, sum = 3
num = 3, sum = 6
…
num = 99, sum = 4950
num = 100, sum = 5050
```

  - Sample **solution**:

```
let sum = 0;
for (let i = 1; i <= 100; i++) {
  sum += i;
  console.log(
    `num = ${i}, sum = ${sum}`);
}
```

Problem: Numbers and Sums

- Write a JavaScript function **printNumsSums(n)** to print the **numbers 1…n**, along with **their cumulative sums**
  - Output for **n** = **6**:

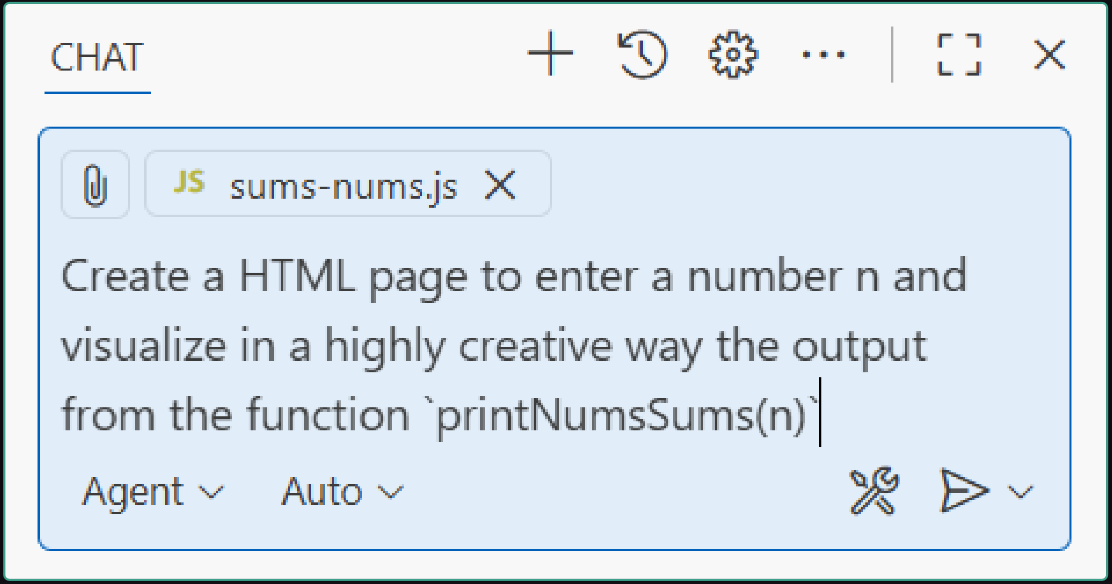
```
num = 1, sum = 1
num = 2, sum = 3
num = 3, sum = 6
num = 4, sum = 10
num = 5, sum = 15
num = 6, sum = 21
```

```javascript
function printNumsSums(n) {
  let sum = 0;
  for (let i = 1; i <= n; i++) {
    sum += i;
    console.log(
      `num = ${i}, sum = ${sum}`);
  }
}
```

Judge link: https://alpha.judge.softuni.org/contests/control-flow-logic/5271

Creative a Visualization of Sums

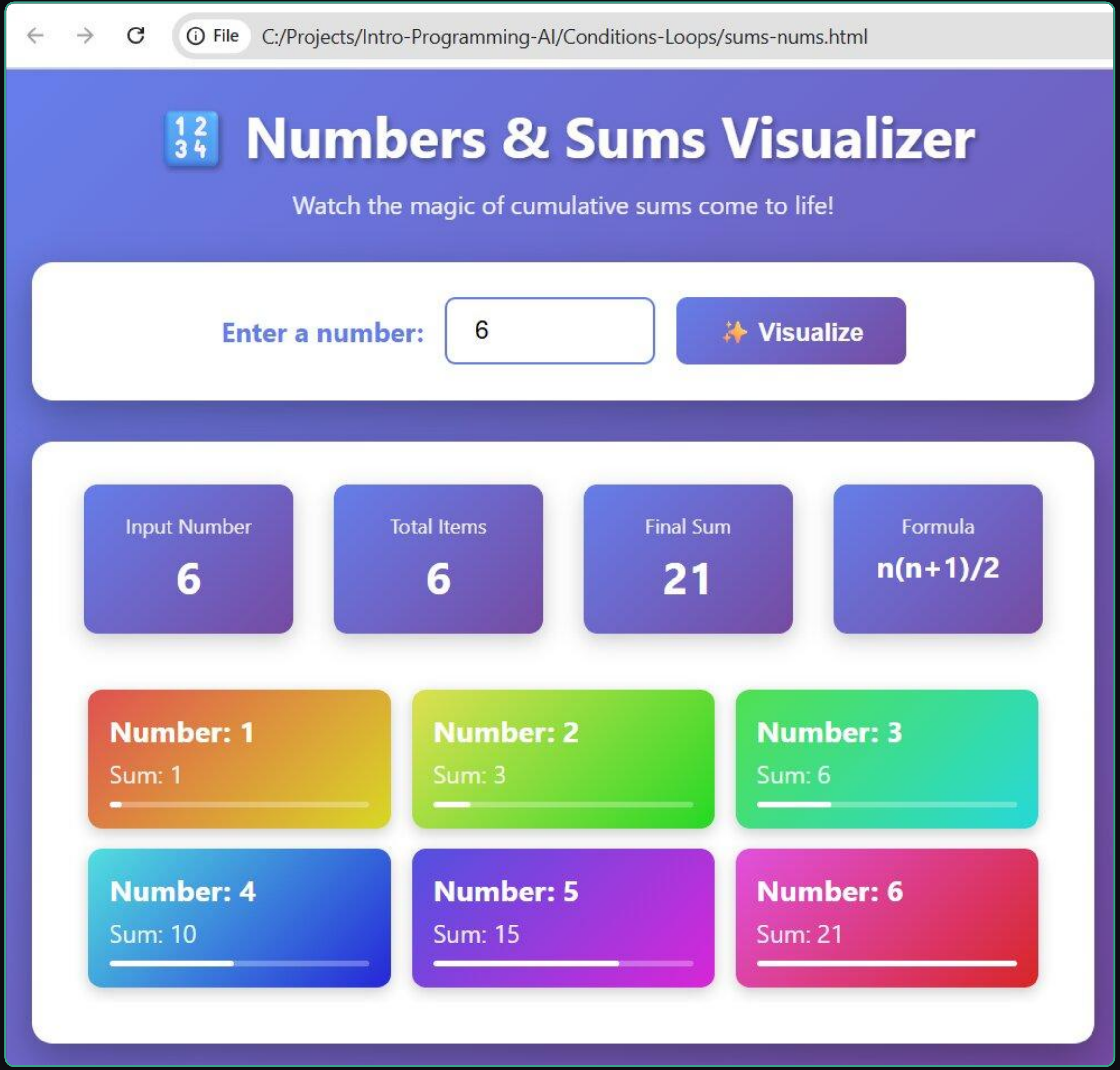
Create a HTML page to enter a number n and visualize in a highly creative way the output from the function `printNumsSums(n)`.


CHAT

sums-nums.js

Create a HTML page to enter a number n and visualize in a highly creative way the output from the function `printNumsSums(n)`

Agent Auto


C:/Projects/Intro-Programming-AI/Conditions-Loops/sums-nums.html

Numbers & Sums Visualizer

Watch the magic of cumulative sums come to life!

Enter a number: 6 ✨ Visualize

Input Number 6

Total Items 6

Final Sum 21

Formula n(n+1)/2

Number: 1 Sum: 1

Number: 2 Sum: 3

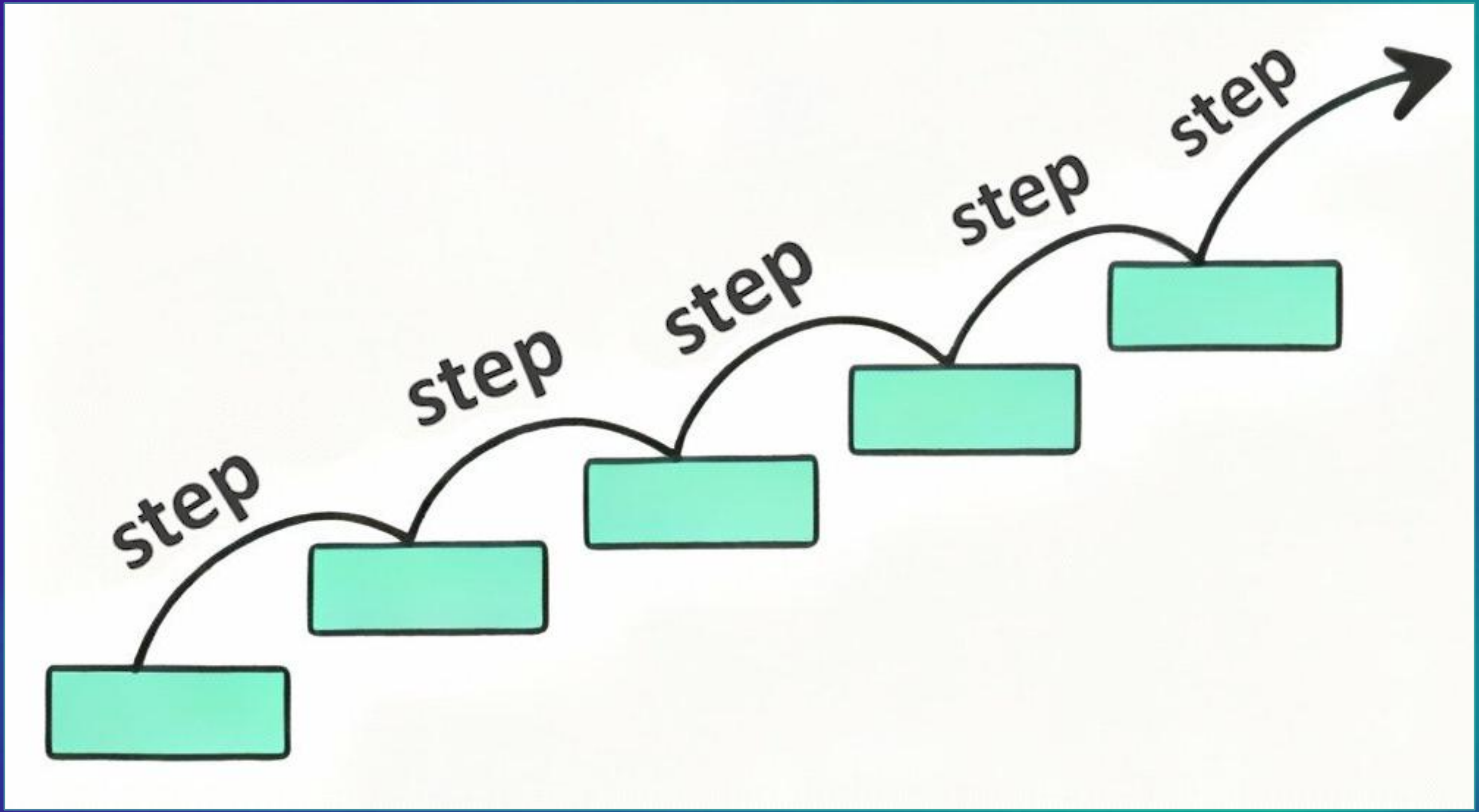
Number: 3 Sum: 6

Number: 4 Sum: 10

Number: 5 Sum: 15

Number: 6 Sum: 21

Loops with a Step



Loops with a Step

- **Backwards iteration** loop: print numbers 20, 19, …, 1

```
for (let i = 20; i >= 1; i--)
  console.log(i);
```

- Loop with a **step +2**: print numbers 100, 102, 104, …, 200

```
for (let i = 100; i <= 200; i += 2)
  console.log(i);
```

- Loop with a **step /2**: print numbers 1024, 512, 256, …, 1

```
for (let i = 1024; i >= 1; i /= 2)
  console.log(i);
```

Problem: Color Gradient Loop

- Write a function to calculate a **color gradient**: from **black** {**0**, **0**, **0**} to given **RGB color** {**red**, **green**, **blue**} in **n** steps

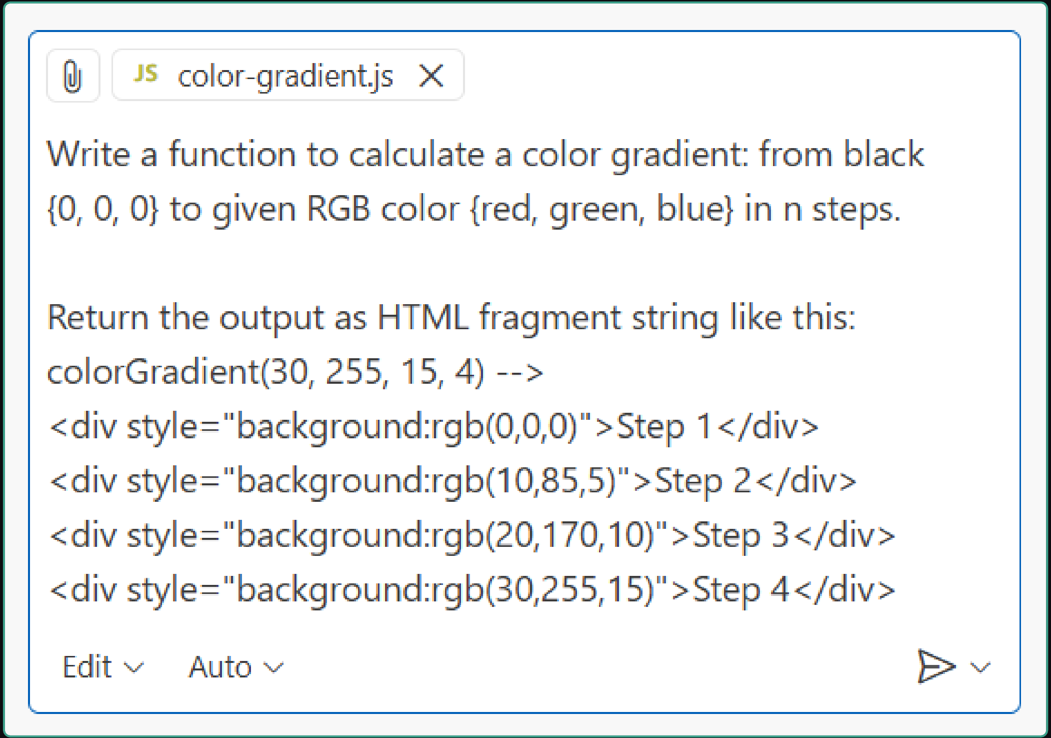
```
function colorGradient(red, green, blue, steps) { … }
```

- Return the output as **HTML fragment** string like this:

```
colorGradient(30, 255, 15, 4)
```

```
<div style="background:rgb(0,0,0)">Step 1</div>
<div style="background:rgb(10,85,5)">Step 2</div>
<div style="background:rgb(20,170,10)">Step 3</div>
<div style="background:rgb(30,255,15)">Step 4</div>
```

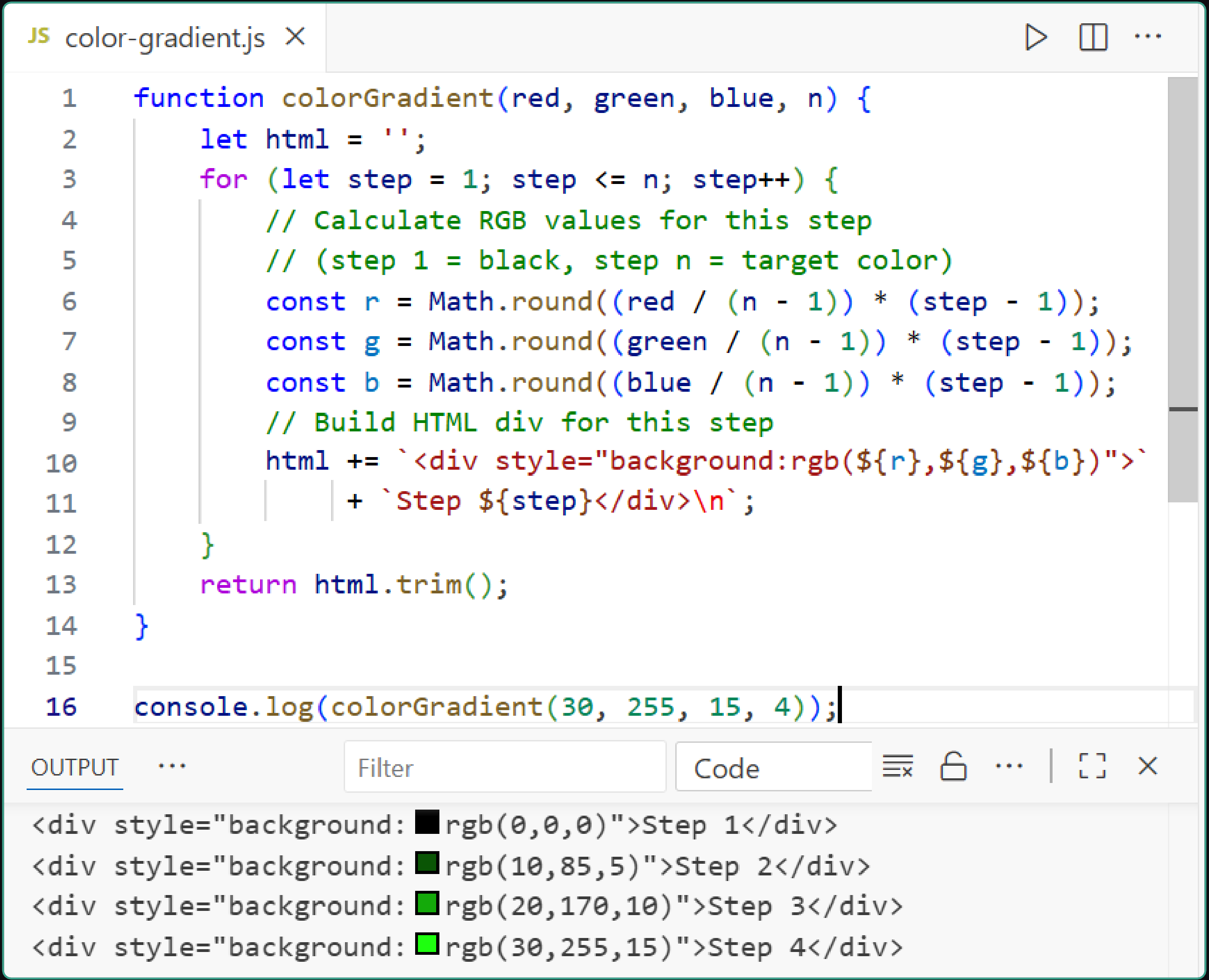
Solution: Color Gradient Loop



color-gradient.js

Write a function to calculate a color gradient: from black {0, 0, 0} to given RGB color {red, green, blue} in n steps.

Return the output as HTML fragment string like this:
colorGradient(30, 255, 15, 4) -->
<div style="background:rgb(0,0,0)">Step 1</div>
<div style="background:rgb(10,85,5)">Step 2</div>
<div style="background:rgb(20,170,10)">Step 3</div>
<div style="background:rgb(30,255,15)">Step 4</div>



color-gradient.js

```javascript
function colorGradient(red, green, blue, n) {
    let html = '';
    for (let step = 1; step <= n; step++) {
        // Calculate RGB values for this step
        // (step 1 = black, step n = target color)
        const r = Math.round((red / (n - 1)) * (step - 1));
        const g = Math.round((green / (n - 1)) * (step - 1));
        const b = Math.round((blue / (n - 1)) * (step - 1));
        // Build HTML div for this step
        html += `<div style="background:rgb(${r},${g},${b})">`
            + `Step ${step}</div>\n`;
    }
    return html.trim();
}

console.log(colorGradient(30, 255, 15, 4));
```

```
<div style="background:■rgb(0,0,0)">Step 1</div>
<div style="background:■rgb(10,85,5)">Step 2</div>
<div style="background:■rgb(20,170,10)">Step 3</div>
<div style="background:■rgb(30,255,15)">Step 4</div>
```

Judge link:
https://alpha.judge.softuni.org/contests/control-flow-logic/5271

Learn to Read and Understand Code!

SoftUni AI

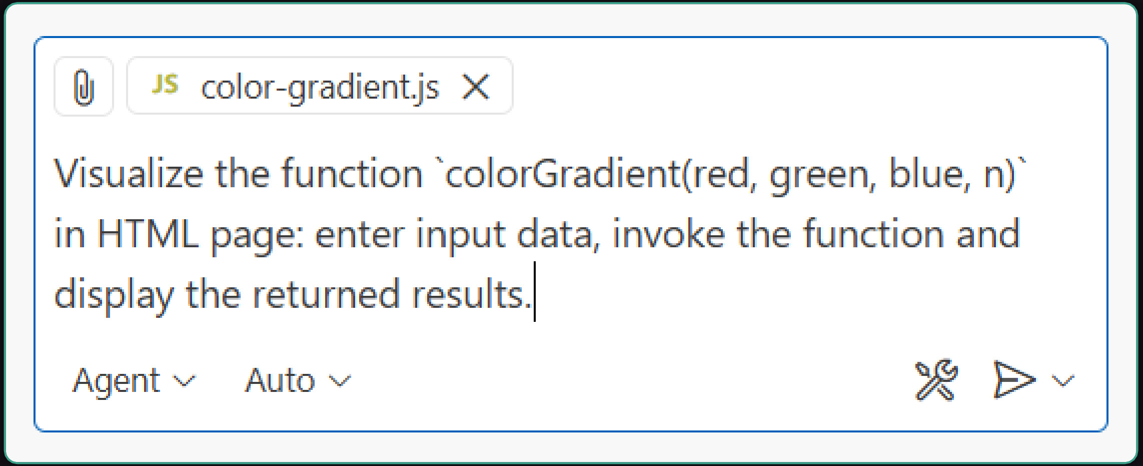
- The code for "color gradient" is not easy for beginners

Learn to read and understand the code, not to write it!

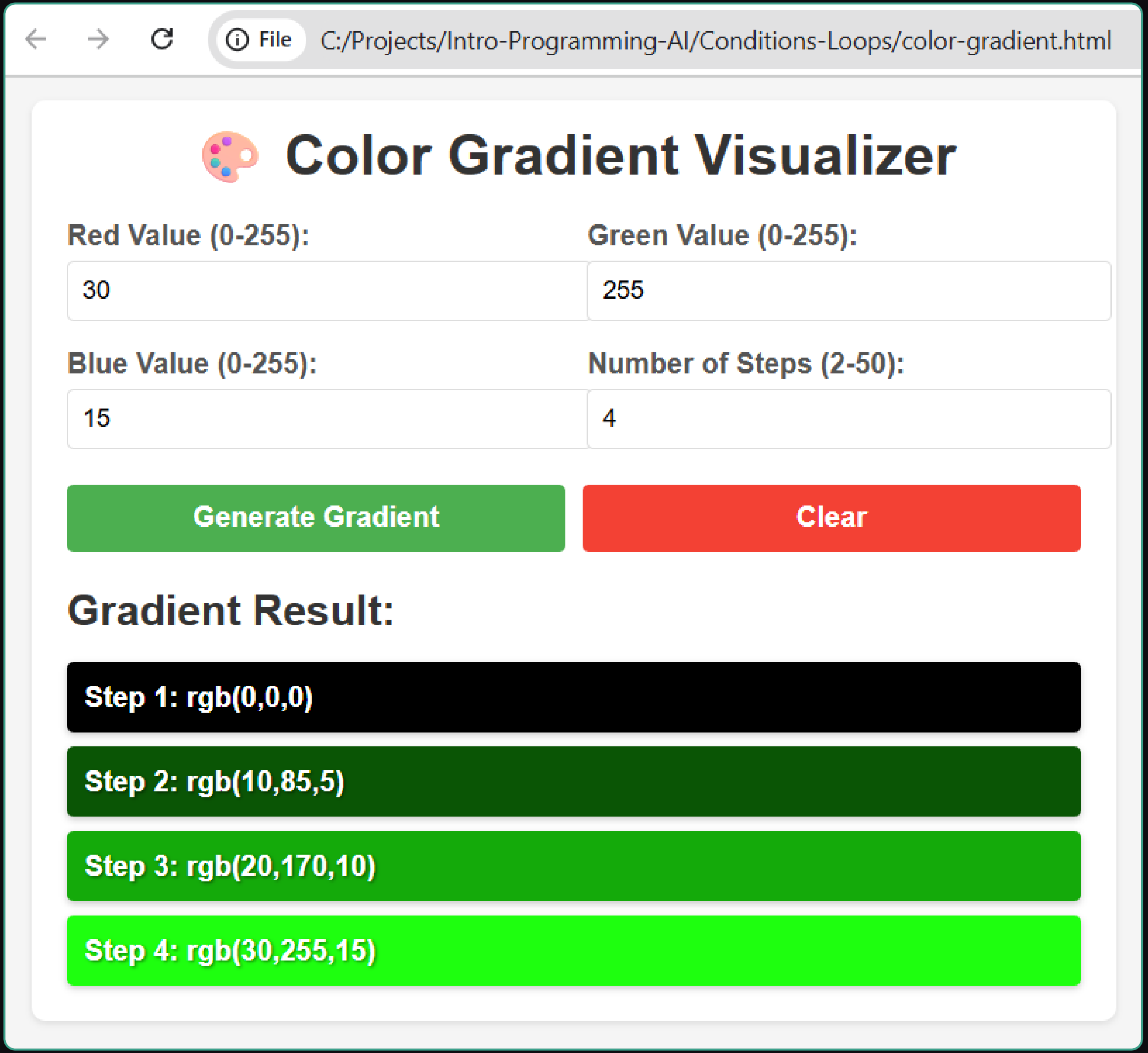
- In the AI era "***writing code***" is no longer a human activity
  - AI agents are **faster** (and **better** in most cases)
- **Reading** and **understanding** the code and **checking** if it works as intended, is still a **human activity**

Visualize the Gradient Loop as HTML

- Run a simple **prompt** in the GitHub Copilot agent:


color-gradient.js

Visualize the function `colorGradient(red, green, blue, n)` in HTML page: enter input data, invoke the function and display the returned results.


C:/Projects/Intro-Programming-AI/Conditions-Loops/color-gradient.html

Color Gradient Visualizer

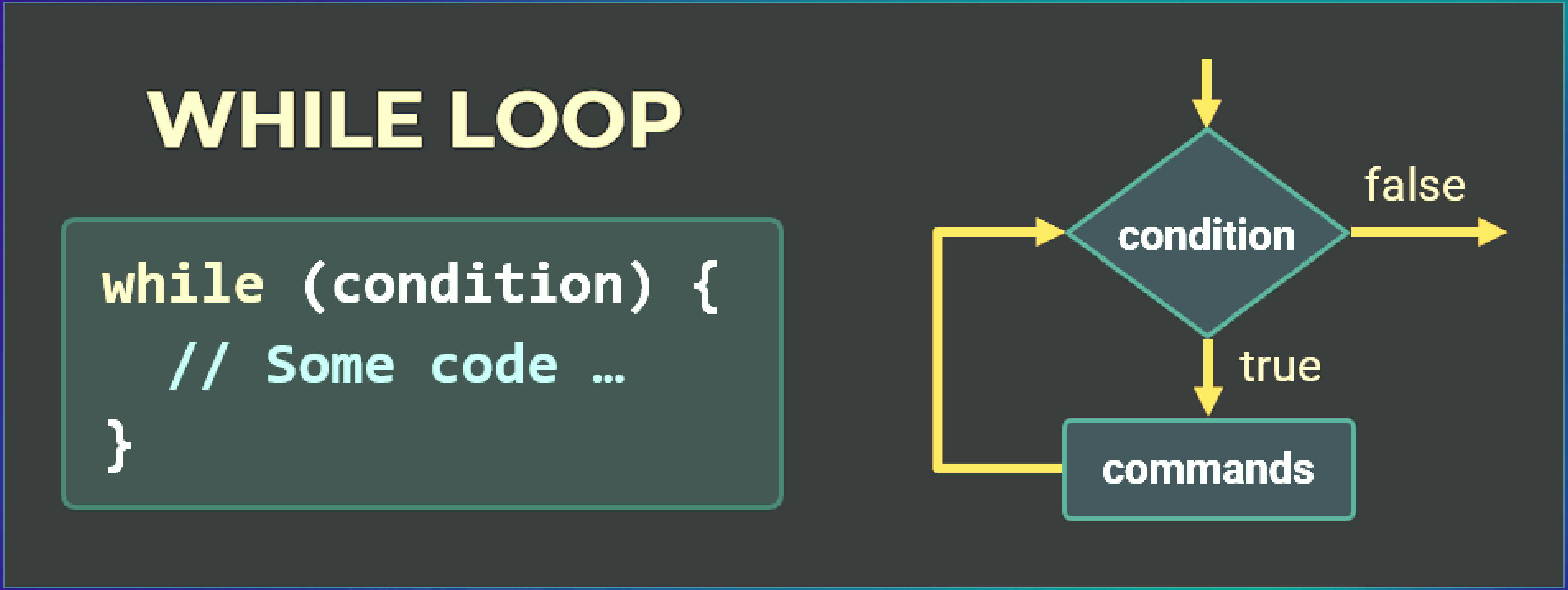
Red Value (0-255): 30
Green Value (0-255): 255
Blue Value (0-255): 15
Number of Steps (2-50): 4

Gradient Result:

```
Step 1: rgb(0,0,0)
Step 2: rgb(10,85,5)
Step 3: rgb(20,170,10)
Step 4: rgb(30,255,15)
```

While Loops

While a Condition Holds, Repeat the Loop Body



WHILE LOOP

```
while (condition) {
  // Some code …
}
```

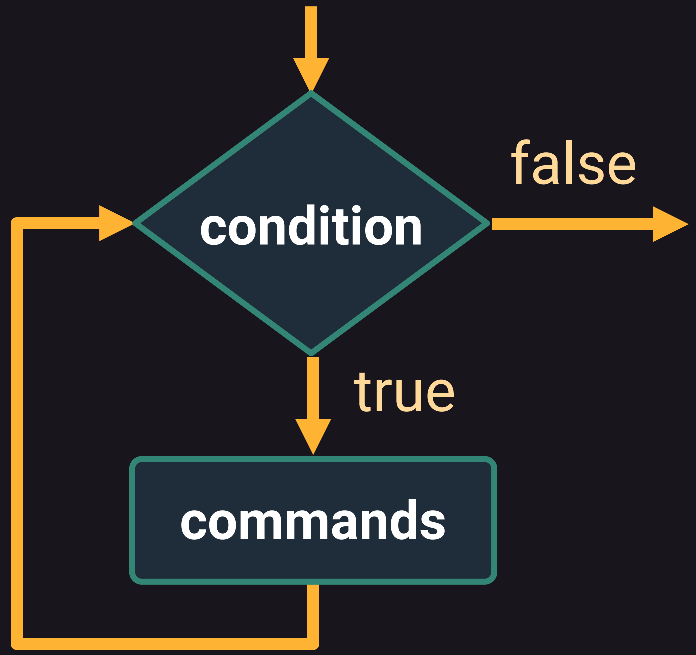
While Loop

- **While-loops** are simple:
  - While given **condition** holds true, **repeat** a block of code

```
while (condition) {
  // Some code …
}
```

- Example:

```
let num = 50;
while (num >= 1) {
  console.log(num);
  num = num / 2;
}
```



Problem: Sequence 2k+1

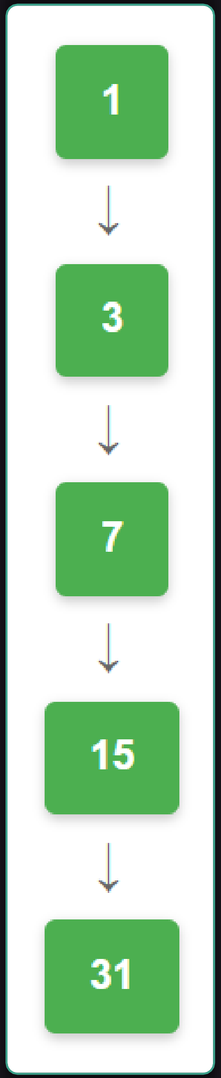
- We are given the sequence: **1**, **3**, **7**, **15**, **31**, …
  - **First** = **1**, each following = **2 * previous + 1**
- Write a function **seq2k1(n)** to return all sequence members **≤ n**, as comma separated string:

```
seq2k1(5)
1, 3
```

```
seq2k1(15)
1, 3, 7, 15
```

```
seq2k1(32)
1, 3, 7, 15, 31
```

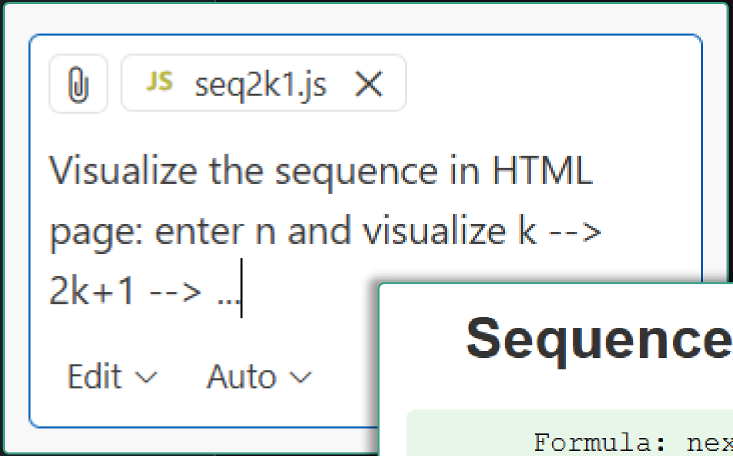
```
seq2k1(3000)
1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047
```



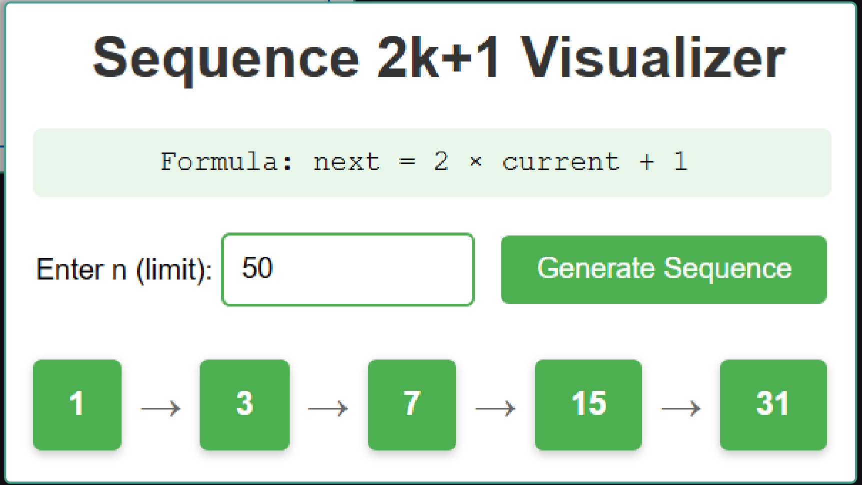
Solution: Sequence 2k+1

```
function seq2k1(n) {
  let seq = 1, output = "";

  while (seq <= n) {
    if (seq > 1)
      output += ", ";
    output += seq;
    seq = 2 * seq + 1;
  }

  return output;
}
```

- Let's **visualize** the sequence in HTML:


seq2k1.js

Visualize the sequence in HTML page: enter n and visualize k --> 2k+1 --> ...


Sequence 2k+1 Visualizer

```
Formula: next = 2 × current + 1
```

Enter n (limit): 50

1 → 3 → 7 → 15 → 31

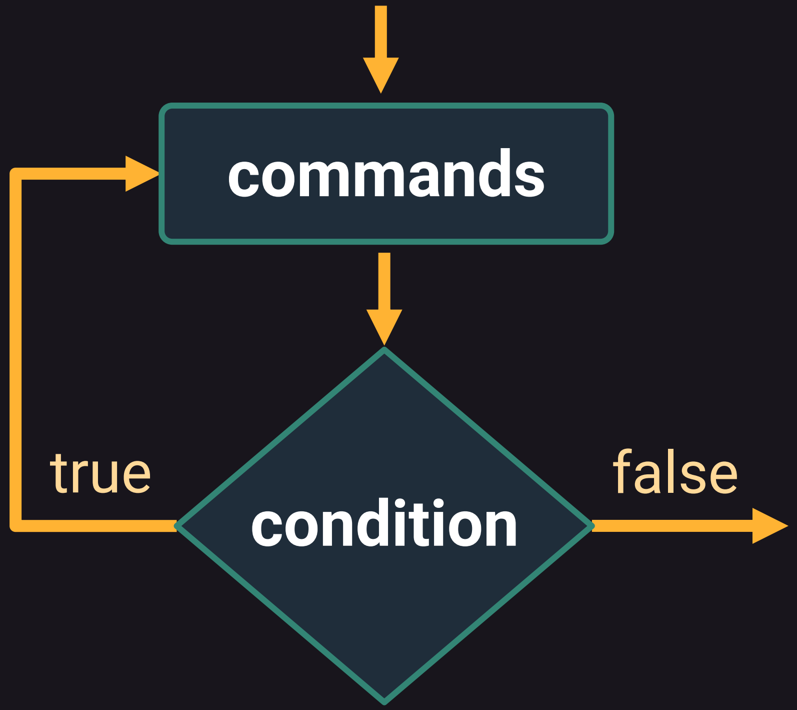
Judge link: https://alpha.judge.softuni.org/contests/control-flow-logic/5271

Do-While Loop

- **Do-while loops** are simple:
  - **Repeat** a block of code while an **exit condition** holds
- Example:

```
let num = 50;
do {
  console.log(num);
  num = num / 2;
} while (num >= 1);
```

```
do {
  // Some code …
} while (condition);
```



Infinite Loops

- **Infinite loops** are loops without an exit condition
  - In most cases are created **by mistake**
- Infinite loop without an exit:

```
let num = 0;
while (true) {
  console.log(++num);
}
```

```
let num = 0;
for (;;) {
  console.log(++num);
}
```

- Infinite loops may cause the app to "***hang***" (to make their **UI unresponsive**)

Infinite Loops with a break

- This loop looks like infinite, but exits after 3 seconds

```
let num = 0;
const startTime = Date.now();

while (true) {
  console.log(++num);
  if (Date.now() - startTime >= 3000)
    break;
}
```

Problem: Guess a Number

- Generate a **random secret number** in the range [1…10] and **interactively guess it**
  - User makes a guess
  - If the guess matches the secret, print "**Correct!**" and **exit**
  - Otherwise, print "**Higher**" or "**Lower**" and try another guess

```
Guess the number (1-10):
5
Higher
Guess the number (1-10):
8
Lower
Guess the number (1-10):
7
Correct!
```

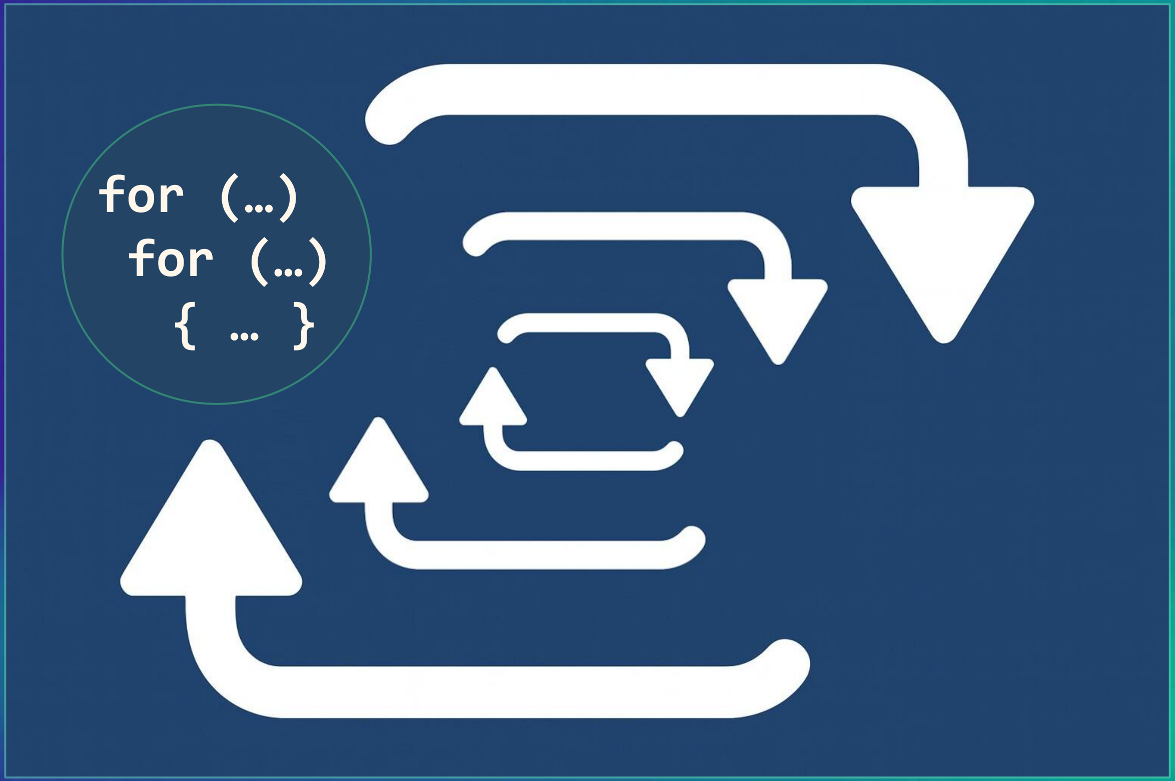
Solution: Guess a Number

```
const secret = Math.floor(
  Math.random() * 10) + 1;
while (true) {
  let guess = Number(prompt(
    "Guess the number (1-10):"));
  if (guess == secret) {
    alert("Correct!");
    break;
  }
  if (guess < secret)
    alert("Higher");
  else
    alert("Lower");
}
```

- **Important**: this code will run **in the browser only**!
  - **prompt()** and **alert()** are unavailable in Node.js
- **Math.random()** returns a random number [0…1)

Nested Loops

Loops inside Other Loops



```
for (…)
 for (…)
   { … }
```

Nested Loops

SoftUni AI

```
for (…)
 for (…)
  { … }
```

- **Nested loop** == a loop inside another loop

```
for (let row = 1; row <= 5; row++) {
  let stars = "";
  for (let cols = 1; cols <= 10; cols++) {
    stars = stars + "*";
  }
  console.log(stars);
}
```

Outer loop

Inner loop

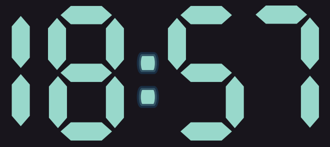
```
**********
**********
**********
**********
**********
```

- This will print a **10 x 5 rectangle of stars**

Example: Time from 0:00 to 23:59

- Print the time from **0:00**, **0:01**, **0:02**, … to **23:59**
  - Print the **hours** (0–23), and for each hour, print the **minutes** (0–59)

```
for (let h = 0; h <= 23; h++) // outer loop
  for (let m = 0; m <= 59; m++) // inner loop
    console.log(`${h}:${m < 10 ? '0' : ''}${m}`);
```



```
0:00    1:00    2:00          22:00    23:00
0:01    1:01    2:01          22:01    23:01
0:02    1:02    2:02    …     22:02    23:02
…       …       …             …        …
0:59    1:59    2:59          22:59    23:59
```

Problem: Time from 0:00 to 23:59

- Write a function to print the clock time
  - From **0:00**, **0:01**, **0:02**, … to **23:59**
- Solution: use **nested loops**
  - Outer loop: **0..23**, inner loop: **0..59**

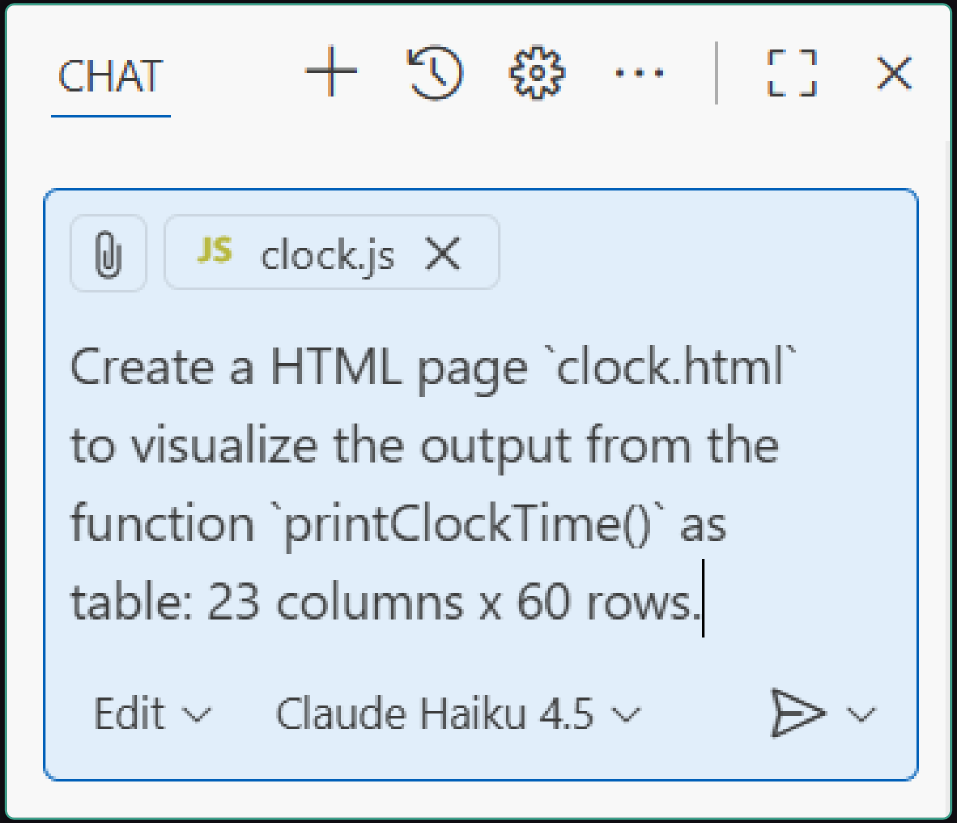
```
for (…)
 for (…)
   { … }
```

```
function printClockTime() {
  for (let h = 0; h <= 23; h++) // outer loop
    for (let m = 0; m <= 59; m++) // inner loop
      console.log(`${h}:${m < 10 ? '0' : ''}${m}`);
}
```

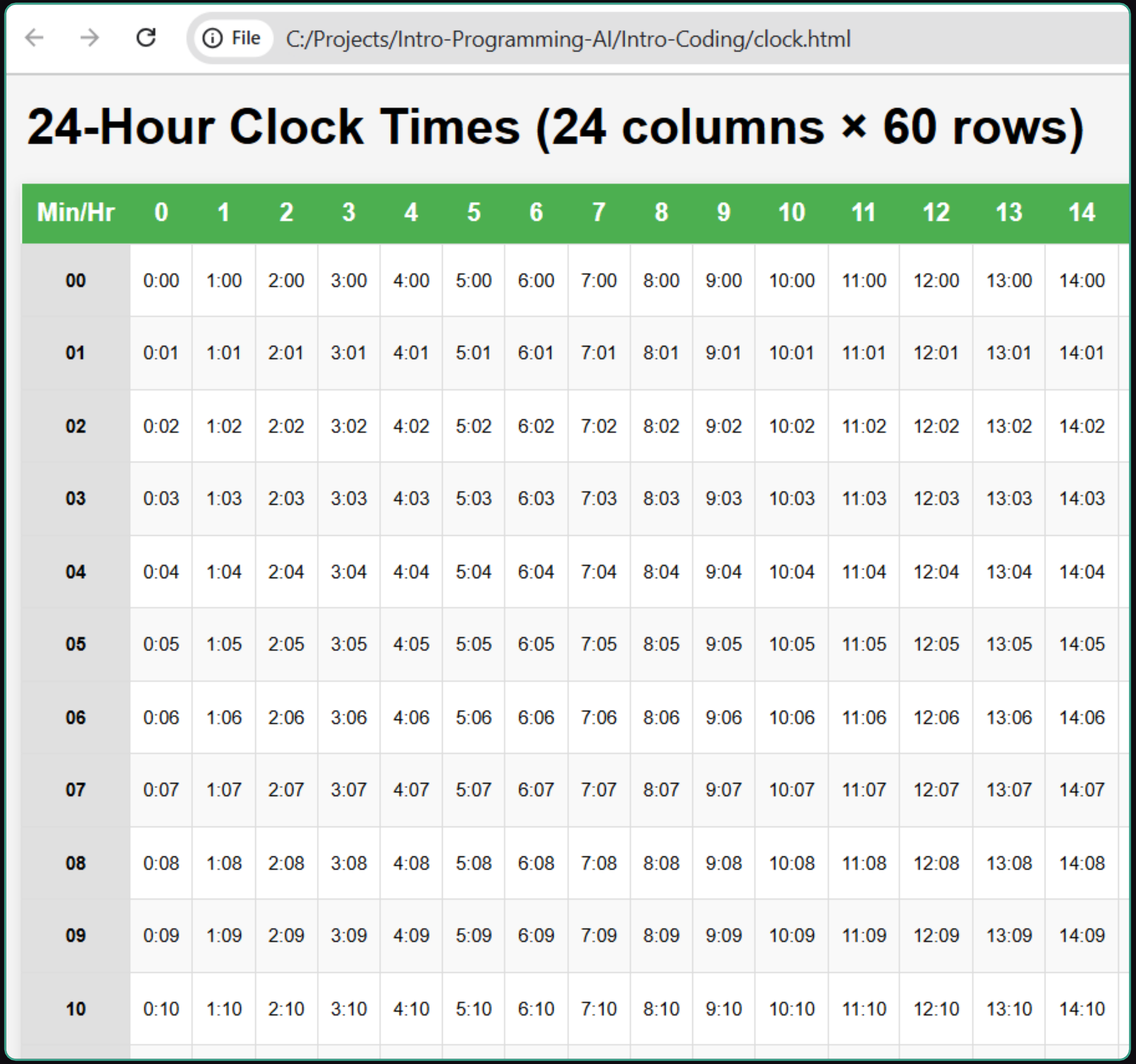
Judge link: https://alpha.judge.softuni.org/contests/control-flow-logic/5271

Visualize the Function with HTML

- Let's **visualize** the output from the function in HTML:



Create a HTML page `clock.html` to visualize the output from the function `printClockTime()` as table: 23 columns x 60 rows.


File C:/Projects/Intro-Programming-AI/Intro-Coding/clock.html

24-Hour Clock Times (24 columns × 60 rows)

Min/Hr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
00	0:00	1:00	2:00	3:00	4:00	5:00	6:00	7:00	8:00	9:00	10:00	11:00	12:00	13:00	14:00
01	0:01	1:01	2:01	3:01	4:01	5:01	6:01	7:01	8:01	9:01	10:01	11:01	12:01	13:01	14:01
02	0:02	1:02	2:02	3:02	4:02	5:02	6:02	7:02	8:02	9:02	10:02	11:02	12:02	13:02	14:02
03	0:03	1:03	2:03	3:03	4:03	5:03	6:03	7:03	8:03	9:03	10:03	11:03	12:03	13:03	14:03
04	0:04	1:04	2:04	3:04	4:04	5:04	6:04	7:04	8:04	9:04	10:04	11:04	12:04	13:04	14:04
05	0:05	1:05	2:05	3:05	4:05	5:05	6:05	7:05	8:05	9:05	10:05	11:05	12:05	13:05	14:05
06	0:06	1:06	2:06	3:06	4:06	5:06	6:06	7:06	8:06	9:06	10:06	11:06	12:06	13:06	14:06
07	0:07	1:07	2:07	3:07	4:07	5:07	6:07	7:07	8:07	9:07	10:07	11:07	12:07	13:07	14:07
08	0:08	1:08	2:08	3:08	4:08	5:08	6:08	7:08	8:08	9:08	10:08	11:08	12:08	13:08	14:08
09	0:09	1:09	2:09	3:09	4:09	5:09	6:09	7:09	8:09	9:09	10:09	11:09	12:09	13:09	14:09
10	0:10	1:10	2:10	3:10	4:10	5:10	6:10	7:10	8:10	9:10	10:10	11:10	12:10	13:10	14:10

Simple Multiplication Table

- Example: print the multiplication table in the format below

```
1 * 1 = 1
1 * 2 = 2
…
1 * 9 = 9
```

```
2 * 1 = 2
2 * 2 = 4
…
2 * 9 = 18
```

…

```
9 * 1 = 9
9 * 2 = 18
…
9 * 9 = 81
```

```
10 * 1 = 10
10 * 2 = 20
…
10 * 10 = 100
```

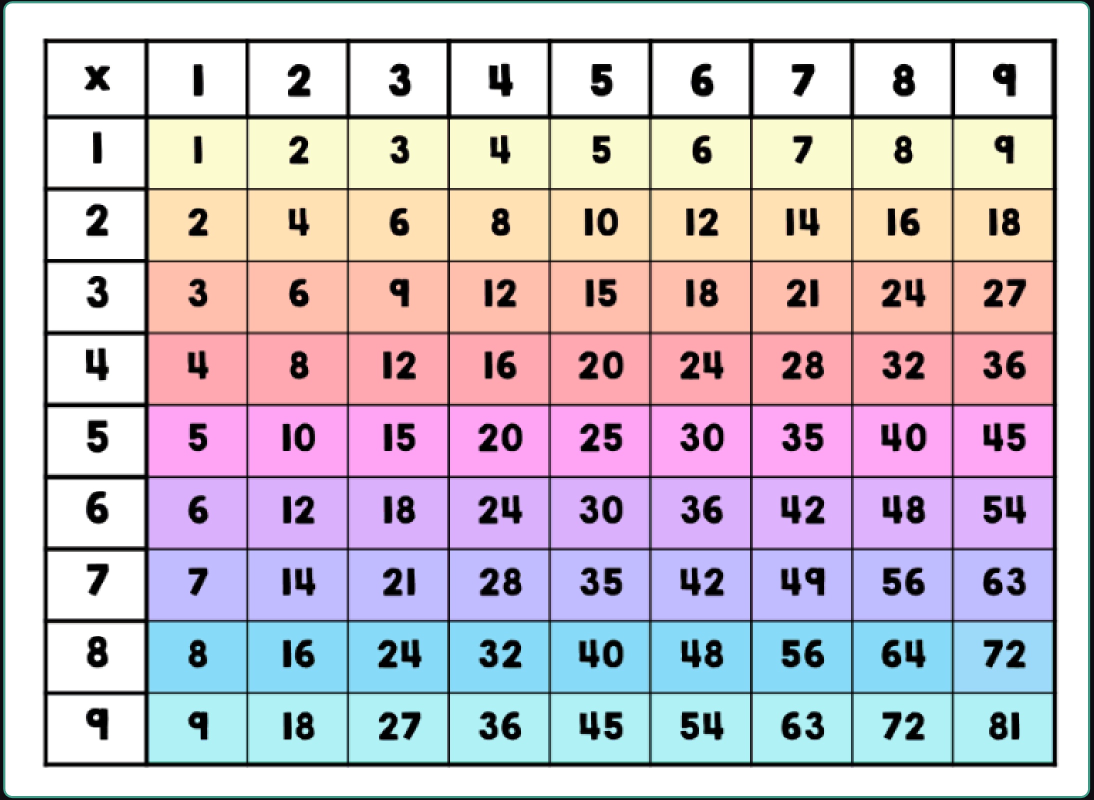
- Use a loop 1…10, holding inner loop 1..10:

```
for (let num1 = 1; num1 <= 10; num1++) // outer loop
  for (let num2 = 1; num2 <= 10; num2++) // nested loop
    console.log(`${num1} * ${num2} = ${num1 * num2}`);
```

Square Multiplication Table 9 x 9

- Now let's print a square multiplication table of size 9 x 9:

```
 x  |  1  2  3  4  5  6  7  8  9
--- |---------------------------
 1  |  1  2  3  4  5  6  7  8  9
 2  |  2  4  6  8 10 12 14 16 18
 3  |  2  4  6  8 10 12 14 16 18
 4  |  4  8 12 16 20 24 28 32 36
 5  |  5 10 15 20 25 30 35 40 45
 6  |  6 12 18 24 30 36 42 48 54
 7  |  7 14 21 28 35 42 49 56 63
 8  |  8 16 24 32 40 48 56 64 72
 9  |  9 18 27 36 45 54 63 72 81
```



x	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

Problem: Multiplication Table N x N

- Write a function to print a multiplication table of size **N** x **N**
  - Use appropriate **column size** to avoid overflow, but be consistent
  - This is how the table should look like:

```
x | 1 2 3
--|-------
1 | 1 2 3
2 | 2 4 6
3 | 2 4 6
```

n = 3, col width = 2,
first col width = 1

```
x |  1  2  3  4
--|-------------
1 |  1  2  3  4
2 |  2  4  6  8
3 |  2  4  6  8
4 |  4  8 12 16
```

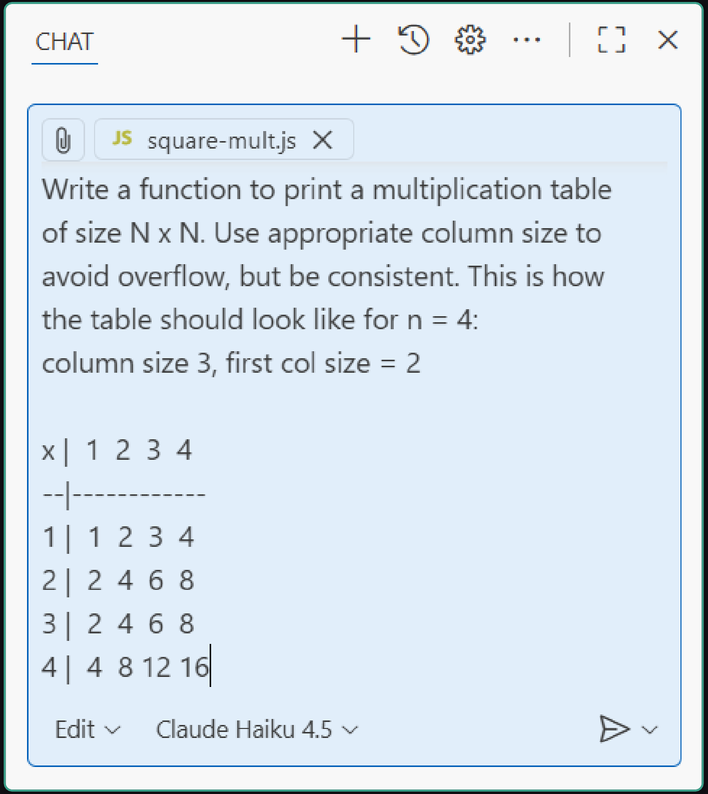
n = 4, col width = 3,
first col width = 1

n = 10, col width = 4,
first col width = 2

```
 x |   1   2   3   4   5   6   7   8   9  10
---|----------------------------------------
 1 |   1   2   3   4   5   6   7   8   9  10
 2 |   2   4   6   8  10  12  14  16  18  20
 3 |   2   4   6   8  10  12  14  16  18  30
 4 |   4   8  12  16  20  24  28  32  36  40
 5 |   5  10  15  20  25  30  35  40  45  50
 6 |   6  12  18  24  30  36  42  48  54  60
 7 |   7  14  21  28  35  42  49  56  63  70
 8 |   8  16  24  32  40  48  56  64  72  80
 9 |   9  18  27  36  45  54  63  72  81  90
10 |  10  20  30  40  50  60  70  80  90 100
```

Solution: Multiplication Table N x N

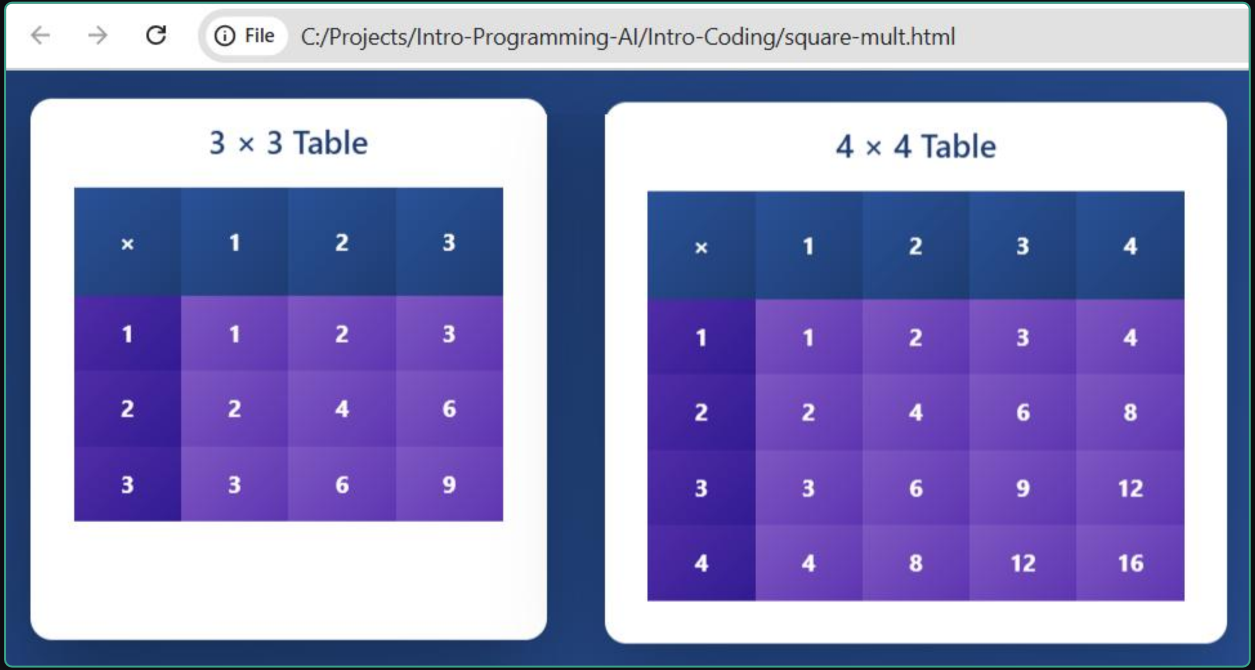
- For some people this could be **too complex** to write by hand, so we shall use a **Copilot prompt**:



Write a function to print a multiplication table of size N x N. Use appropriate column size to avoid overflow, but be consistent. This is how the table should look like for n = 4:
column size 3, first col size = 2

```
x|  1  2  3  4
--|------------
1|  1  2  3  4
2|  2  4  6  8
3|  2  4  6  8
4|  4  8 12 16
```

- Create a colorful HTML visualization


File C:/Projects/Intro-Programming-AI/Intro-Coding/square-mult.html

3 × 3 Table

×	1	2	3
1	1	2	3
2	2	4	6
3	3	6	9

4 × 4 Table

×	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12
4	4	8	12	16

Judge link: https://alpha.judge.softuni.org/contests/control-flow-logic/5271

Problem: Triples

- Write a function **triples(n)** to print **all triples {a, b, c}** out of the numbers [**1…n**], in increasing order

```
triples(3)
1 2 3
```

```
triples(4)
1 2 3
1 2 4
1 3 4
2 3 4
```

```
triples(5)
1 2 3    1 4 5
1 2 4    2 3 4
1 2 5    2 3 5
1 3 4    2 4 5
1 3 5    3 4 5
```

Solution: Triples

- We can **nest loops several times**
- Example: triple nested loops

```
function triples(n) {
  for (let a = 1; a <= n; a++) // outer loop
    for (let b = a+1; b <= n; b++) // middle loop
      for (let c = b+1; c <= n; c++) // inner loop
        console.log(`${a} ${b} ${c}`);
}
```

Judge link: https://alpha.judge.softuni.org/contests/control-flow-logic/5271

Nested Loops with If-Else

Implementing More Complex Logic

Problem: Time Range

- Write a function to print the clock times in given **time range**
  - Use **nested loops** for the hours and minutes

```
timeRange(10, 58, 11, 2)
```

```
10:58
10:59
11:00
11:01
11:02
```

```
timeRange(7, 0, 7, 4)
```

```
7:00
7:01
7:02
7:03
7:04
```

Solution: Time Range (Wrong)

```
function timeRange(startHour, startMins, endHour, endMins) {
  for (let h = startHour; h <= endHour; h++) {
    for (let m = startMins; m <= endMins; m++) {
      console.log(`${h}:${m < 10 ? '0' : ''}${m}`);
    }
  }
}
```

- What will be the result of `printTimeRange(7, 55, 9, 15)`?

Judge link: https://alpha.judge.softuni.org/contests/control-flow-logic/5271

Solution: Time Range (Wrong Again)

```
function timeRange(startHour, startMins, endHour, endMins) {
  for (let h = startHour; h <= endHour; h++) {
    let currentStartMins = startMins;
    if (h > startHour) currentStartMins = 0;
    let currentEndMins = endMins;
    if (h < endHour) currentEndMins = 59;
    for (let m = currentStartMins; m <= currentEndMins; m++) {
      console.log(`${h}:${m < 10 ? '0' : ''}${m}`);
    }
  }
}
```

What will be the result of **printTimeRange(23, 58, 0, 2)**?

Judge link: https://alpha.judge.softuni.org/contests/control-flow-logic/5271

Solution: Time Range (Correct)

```
function timeRange(startHour, startMins, endHour, endMins) {
  if (startHour > endHour) endHour += 24;
  for (let h = startHour; h <= endHour; h++) {
    let currentStartMins = startMins;
    if (h > startHour) currentStartMins = 0;
    let currentEndMins = endMins;
    if (h < endHour) currentEndMins = 59;
    for (let m = currentStartMins; m <= currentEndMins; m++) {
      console.log(`${h % 24}:${m < 10 ? '0' : ''}${m}`);
    }
  }
}
```

Judge link: https://alpha.judge.softuni.org/contests/control-flow-logic/5271

Lesson Summary

- **If-else** statements implement conditional logic
  - Run different code blocks depending on input conditions
  - Conditions can be **complex**, if-else can be **nested**
  - Switch-case is like long if-else chain
- **Loops** repeat a block of code multiple times
  - Pass through ranges of values, e.g. **for i = 1 … 10**
  - Run some code **while** certain condition holds true
- **Nested loops** are loops inside other loops

Questions?

Postbank – Exclusive Partner for SoftUni AI

- One of the leading **banking institutions** in Bulgaria
- Member of the Eurobank Group with € 99.6 billion of assets
- Innovative trendsetter in the digitalization and transformation
- Certified Top Employer 2024 by the international Top Employers Institute
- AI integration for business, learning and development
- AI Assistants for talents: Story Builder, CV Assistant, Interview Trainer
- Proven people care and wellbeing initiatives
- Benefits and unlimited access to professional, personal and leadership trainings and programs
- www.postbank.bg / careers.postbank.bg

Postbank

top EMPLOYER БЪЛГАРИЯ BULGARIA 2024
CERTIFIED EXCELLENCE IN EMPLOYEE CONDITIONS

Diamond Partners of Software University

Software University

Coca-Cola HBC Bulgaria

Postbank

top EMPLOYER България Bulgaria 2025 FOR A BETTER WORLD OF WORK

SUPER HOSTING .BG

encorp.ai

createX

DRAFT KINGS

THE CROWN IS YOURS

INDEAVR

Serving the high achievers

VIVACOM

Diamond Partners of SoftUni Digital

SoftUni Digital

Coca-Cola HBC Bulgaria

Postbank

top EMPLOYER България Bulgaria 2025 FOR A BETTER WORLD OF WORK

SUPER HOSTING .BG

encorp.ai

DRAFT KINGS

THE CROWN IS YOURS

createX

INDEAVR

Serving the high achievers

VIVACOM

Diamond Partners of SoftUni Digital

SoftUni Digital

HUMAN

IMPULSE MEDIA®

1 FOR FIT

NETPEAK

DIGITAL GROWTH PARTNER

ABC DESIGN & COMMUNICATION

Zahara

dig.it.all

ETIENYANEV

Break Your Limits. Live Your Brand.

Aleikov studio

Diamond Partners of SoftUni Creative

SoftUni Creative

Coca-Cola HBC Bulgaria

Postbank

top EMPLOYER България Bulgaria 2025 FOR A BETTER WORLD OF WORK

SUPER HOSTING .BG

encorp.ai

DRAFT KINGS THE CROWN IS YOURS

createX

INDEAVR Serving the high achievers

VIVACOM

Organization Partners of SoftUni Creative

SoftUni Creative

dequitas.

Фигмайстор

Studio π
Think Big • Design Smart

дизайнът на нещата

REDV® PRINT STUDIO

THE BUCKS TOWN'S WORK